In the claims:

Following is a complete set of claims as amended with this Response.

1. (Currently Amended) A method comprising:

reading a line of data from a file containing source code written in a high level language;

generating a stream of tokens from said line of data, said stream of tokens representing any of a specific type of macro in said line of data as being expanded while other types of macros are not expanded;

generating a token object for each token, said token object including a visibility variable to represent whether a parser and an output module may view the respective token;

parsing said stream of tokens using a parser and with reference to respective token objects;

inserting commands representing operations to be performed by a macro into said stream of tokens if a macro is present; and

~~writing expanded macro tokens to an output file if said macro is of said specific type of macro; and~~

~~writing an original macro call to said output file if said macro is not said specific type of macro~~

writing said stream of tokens to an output file using an output module and with reference to respective token objects.

2. (Original) The method of claim 1, wherein said generating a stream of tokens further comprises:

determining whether tokens are present in either an input file, a lookahead buffer, or a macro expansion list; and

responsive to finding tokens, reading said tokens first from said lookahead buffer, then from said macro expansion list, then from said input file;

presenting said tokens to a parser so that any macro in said line of data appears to have been expanded.

3. (Previously Presented) The method of claim 1, wherein said parsing further comprises:

reading a token;

determining a type of said read token;

responsive to determining that said read token is an end-of-line, processing an input line of tokens;

responsive to determining that said read token is not a symbol, adding said read token to a current line token list;

responsive to determining that said read token is a symbol that indicates a beginning of a macro definition, recording a macro name and said macro definition and adding said read token to a lookahead buffer; and

responsive to determining that said read token is a symbol that does not indicate a beginning of a macro definition, adding said read token to a current line token list.

4. (Cancelled)

5. (Original) The method of claim 1, wherein said source code written in a high level language comprises a hardware description language (HDL) for representing hardware designs.

6. (Original) The method of claim 1, wherein said specific type of macro comprises a scan macro.

7. (Currently Amended) A method of scan insertion comprising:

reading a hardware description language (HDL) representation of a hardware design, the HDL including a plurality of macro definitions some of which relate to scan insertion;

creating a token stream based on the HDL representation that includes multifaceted tokens that can be hidden from or made visible to a subsequent parsing process by expanding the plurality of macro definitions and making tokens associated with scan macros visible to the subsequent parsing process and marking other tokens as hidden, said multifaceted tokens being associated with a token object for each token, said token object including a visibility variable to represent whether a parser and an output module may view the respective token and a scan variable to represent whether the respective token is related to scan;

performing scan insertion using a parser by parsing those of the multifaceted tokens that are visible to said ~~the~~ parser based on said visibility variable and adding appropriate scan commands; and

using an output module to generate ~~generating~~ a scan inserted HDL file containing expanded versions of the macro definitions which are visible to the output module based on said visibility variable and which relate to scan insertion but that omits expanded versions of those that do not relate to scan insertion based on said scan variable.

8. (Original) The method of claim 7, wherein said HDL comprises a high-level language.

9. (Original) The method of claim 7, wherein said hardware design represents an integrated circuit design.

10. (Currently Amended) A system comprising:

a storage device having stored therein one or more routines for selectively expanding macros within source code; and

a processor coupled to the storage device for executing the one or more routines for selectively expanding macros within source code which, when executing said routine:

reads a line of data from a file containing source code written in a high level language;

generates a stream of tokens from said line of data, said stream of tokens representing any of a specific type of macro in said line of data as being expanded while other types of macros are not expanded;

generates a token object for each token, said token object including a visibility variable to represent whether a parser and an output module may view the respective token;

parses said stream of tokens using a parser and with reference to respective token objects;

inserts commands representing operations to be performed by a macro into said stream of tokens if a macro is present; and

~~writes expanded macro tokens to an output file if said macro is of said specific type of macro; and~~

~~writes an original macro call to said output file if said macro is not said specific type of macro.~~ writes said stream of tokens to an output file using an output module and with reference to respective token objects.

11. (Original) The system of claim 10, wherein said generating a stream of tokens further comprises:

determining whether tokens are present in either an input file, a lookahead buffer, or a macro expansion list; and

responsive to finding tokens, reading said tokens first from said lookahead buffer, then from said macro expansion list, then from said input file;

presenting said tokens to a parser so that any macro in said line of data appears to have been expanded.

12. (Previously Presented) The system of claim 10, wherein said parsing further comprises:

reading a token;

determining a type of said read token;

responsive to determining that said read token is an end-of-line, processing an input line of tokens;

responsive to determining that said read token is not a symbol, adding said read token to a current line token list;

responsive to determining that said read token is a symbol that indicates a beginning of a macro definition, recording a macro name and said macro definition and adding said read token to a lookahead buffer; and

responsive to determining that said read token is a symbol that does not indicate a beginning of a macro definition, adding said read token to a current line token list.

13. (Canceled)

14. (Original) The system of claim 10, wherein said source code written in a high level language comprises a hardware description language (HDL) for representing hardware designs.

15. (Original) The system of claim 10, wherein said specific type of macro comprises a scan macro.

16. (Currently Amended) A machine-readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to selectively expand macros by:

reading a line of data from a file containing source code written in a high level language;

generating a stream of tokens from said line of , said stream of tokens representing any of a specific type of macro in said line of data as being expanded while other types of macros are not expanded;

generating a token object for each token, said token object including a visibility variable to represent whether a parser and an output module may view the respective token;

parsing said stream of tokens using a parser and with reference to respective token objects;

inserting commands representing operations to be performed by a macro into said stream of tokens if a macro is present; and

~~writing expanded macro tokens to an output file if said macro is of said specific type of macro; and~~

~~writing an original macro call to said output file if said macro is not said specific type of macro.~~ writing said stream of tokens to an output file using an output module and with reference to respective token objects.

17. (Original) The machine-readable medium of claim 16, wherein said generating a stream of tokens further comprises:

determining whether tokens are present in either an input file, a lookahead buffer, or a macro expansion list; and

responsive to finding tokens, reading said tokens first from said lookahead buffer, then from said macro expansion list, then from said input file;

presenting said tokens to a parser so that any macro in said line of data appears to have been expanded.

18. (Previously Presented) The machine-readable medium of claim 16, wherein said parsing further comprises:

reading a token;

determining a type of said read token;

responsive to determining that said read token is an end-of-line, processing an input line of tokens;

responsive to determining that said read token is not a symbol, adding said read token to a current line token list;

responsive to determining that said read token is a symbol that indicates a beginning of a macro definition, recording a macro name and macro definition and adding said read token to a lookahead buffer; and

responsive to determining that said read token is a symbol that does not indicate a beginning of a macro definition, adding said read token to a current line token list.

19. (Cancelled)

20. (Original) The machine-readable medium of claim 16, wherein said source code written in a high level language comprises a hardware description language (HDL) for representing hardware designs.

21. (Original) The machine-readable medium of claim 16, wherein said specific type of macro comprises a scan macro.

22. (Currently Amended) A machine-readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to perform scan insertion by:

reading a hardware description language (HDL) representation of a hardware design, the HDL including a plurality of macro definitions some of which relate to scan insertion;

creating a token stream based on the HDL representation that includes multifaceted tokens that can be hidden from or made visible to a subsequent parsing process by expanding the plurality of macro definitions and making tokens associated with scan macros visible to the subsequent parsing process and marking other tokens as hidden, said multifaceted tokens being associated with a token object for each token, said token object including a visibility variable to represent whether a parser and an output module may view the respective token and a scan variable to represent whether the respective token is related to scan;

performing scan insertion using a parser by parsing those of the multifaceted tokens that are visible to said ~~the~~ parser based on the visibility variable and adding appropriate scan commands; and

using an output module to generate ~~generating~~ a scan inserted HDL file containing expanded versions of the macro definitions which are visible to the output module based on said visibility variable and which relate to scan insertion but that omits expanded versions of those that do not relate to scan insertion based on said scan variable.

23. (Original) The machine-readable medium of claim 22, wherein said HDL comprises a high-level language.

24. (Original) The machine-readable medium of claim 22, wherein said hardware design represents an integrated circuit design.

25. (New) The method of claim 1, wherein said writing comprises:

writing expanded macro tokens to said output file if said macro is of said specific type of macro; and

writing an original macro call to said output file if said macro is not said specific type of macro.

26. (New) The machine-readable medium of claim 16, wherein said writing comprises:

writing expanded macro tokens to said output file if said macro is of said specific type of macro; and

writing an original macro call to said output file if said macro is not said specific type of macro.